Lifting Factorization-Based Discrete Wavelet Transform Architecture Design

Wenqing Jiang and Antonio Ortega

***Abstract*—In this paper, two new system architectures, *overlap-state* sequential and *split-and-merge* parallel, are proposed based on a novel *boundary postprocessing* technique for the computation of the discrete wavelet transform (DWT). The basic idea is to introduce multilevel partial computations for samples near data boundaries based on a *finite state machine* model of the DWT derived from the lifting scheme. The key observation is that these partially computed (lifted) results can also be stored back to their original locations and the transform can be continued anytime later as long as these partial computed results are preserved. It is shown that such an extension of the in-place calculation feature of the original lifting algorithm greatly helps to reduce the extra buffer and communication overheads, in sequential and parallel system implementations, respectively. Performance analysis and experimental results show that, for the Daubechies (9,7) wavelet filters, using the proposed *boundary postprocessing* technique, the minimal required buffer size in the line-based sequential DWT algorithm [1] is 40% less than the best available approach. In the parallel DWT algorithm we show 30% faster performance than existing approaches.**

***Index Terms*—Boundary postprocessing, discrete wavelet transform, overlap-state, parallel algorithm, sequential algorithm, split-and-merge.**

I. Introduction

EFFICIENT system architecture design for the discrete wavelet transform (DWT) has received a lot of attention recently [2]–[4], [1], [5]–[8] due to the success of DWT-based techniques in areas as diverse as signal processing, digital communications, numerical analysis, computer vision and computer graphics [9]. Two important parameters have been used to measure the efficiency of practical DWT system designs: 1) the *memory* necessary for the DWT computation (mostly in sequential algorithms) and 2) the *communication* overhead required by parallel DWT algorithms. As a matter of fact, *memory efficiency* is one major design factor for wavelet-based image compression applications in printers, digital cameras and space-borne instruments where large size memory leads to high cost and demands more chip design area [1], [10], [11]. Similarly, *communication efficiency* is critical to the success of parallel DWT systems built upon the network of workstations (NOWs) or local area multicomputers (LAMs), since in these systems cheap but slower communication links are used (as compared with dedicated parallel systems) [12], [8], [13], [14].

The major difficulty in achieving an efficient DWT architecture design (both in terms of memory and communication) is that, with the exception of trivial Haar filters, the DWT is not a block transform. When data has to be processed one block (or one image scanline) at a time in sequential systems [1], [6] or partitioned over multiple processors in parallel systems [4], [8], correct DWT computation near data boundaries requires extra buffer and/or extra communication compared to that needed for a block transform such as the discrete cosine transform (DCT). In standard FFT-based filtering approaches, such a boundary issue can be easily handled with appropriate data overlapping (e.g., the *overlap-save* or *overlap-add* approaches [15]). However, because the DWT consists of *recursive* filtering operations on multilevel *downsampled* data sequences, direct application of the overlapping techniques can be very costly in terms of memory and/or inter-processor communication.

Consider, for example, a J-level wavelet decomposition of a N-point input sequence to be performed using two processors (assuming N is even for simplicity). In this case, either the two processors are given sufficient overlapped data to carry on the whole computation without communicating with each other, or alternatively, they have to communicate samples after each level of the decomposition has been computed. The first approach, *overlapping*, requires that input data near the block boundaries be given to both processors. Since each processor has to compute its own transform for multiple decomposition levels, this overlap can be quite large. As given in the analysis of the spatially segmented wavelet transform (SSWT) by Kossentini [16], the buffer size for a J-level decomposition of a N-point sequence is $N+(2^J-1)(L-2)$ (L is the filter length). As one can see, the overlap increases exponentially with the increase of decomposition level J, which can become significant if long wavelet filters are used and the number of levels of decomposition is large. Notice that the in-place lifting algorithm [17] is already assumed to be used in our work and the focus of this paper is on the reduction of memory at boundaries below the level, $(2^J-1)(L-2)$, required in a standard lifting approach. To the best of our knowledge, reduction of boundary memory has not been addressed within a lifting framework until recently.

As one alternative, the *overlapping* technique has been also applied at each decomposition level rather than once for all as in SSWT. Examples of this approach include the recursive pyramid algorithm (RPA) by Vishwanath [3], and the reduced line-based compression system by Chrysafis *et al.* [1], [10]. These approaches reduce significantly the buffer size to $N+J(L-2)$ for a J-level decomposition of a N-point sequence. Unfortunately, this is still a quite large overlap for some applications, for example, the line-based wavelet image compression system

Manuscript received November 9, 1999; revised September 26, 2000. This research was supported in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, under Cooperative Agreement EEC-9529152. This paper was recommended by Associate Editor Z. Xiong.

W. Jiang was with the Integrated Media System Center, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089 USA. He is now with C-Cube MicroSystems Inc., Milpitas, CA 95035 USA (e-mail: wjiang@c-cube.com).

A. Ortega is with the Integrated Media System Center, Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089 USA (e-mail: ortega@sipi.usc.edu).

Publisher Item Identifier S 1051-8215(01)03818-6.

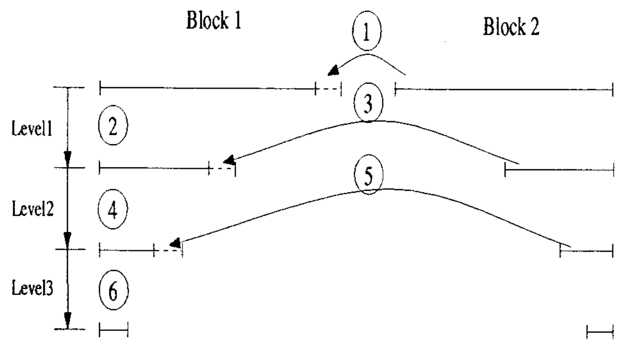


Fig. 1. Example dataflow chart of a three-level wavelet decomposition. Solid lines: completely transformed data. Dashed lines: boundary samples from the neighboring block. Operations 1,3,5: communicate boundary data samples to neighboring blocks. Operations 2,4,6: transform for current level.

described in [1], [10]. In that system, image lines are stored in memory only while they are used to generate output coefficients and are released from memory when no longer needed. This leads to $N = 0$ and the amount of memory is $2(1-2^{-J})(L-2)$ image lines (due to line downsampling) at each stage. Consider a color image of size 4096×4096, such that each color component sample is stored as a 4 bytes floating point number for DWT computation. In this case, one image scanline requires 48 kB. Using the Daubechies (9,7) wavelet filterbank ($L = 9$), for a three-level decomposition, the total memory would be 588 kB. In this paper, we propose a novel technique which can help to reduce the memory to only 296 kB.

In the second approach, *nonoverlapping,*to parallel DWT implementation, input data is not overlapped so the memory requirement is relaxed. But boundary samples need to be exchanged at each decomposition level. Such an approach is used, for example, in the design of mesh and hypercube parallel DWT architectures by Fridman *et al.* [4]. Their analysis shows that, for a J-level wavelet decomposition, J data exchanges are needed between neighboring processors [4] (see Fig. 1 for a three-level example). In order to reduce the communication overhead, Yang *et al.* [12] proposed to use boundary extensions in their DWT system configured from a cluster of SGI workstations. This, however, computes incorrect wavelet coefficients near data boundaries, which causes performance degradation in low-bit rate image coding applications [18].

This provides the motivation to study the problem of block-based DWT computation and its implications on memory and communication overhead in practical system designs. In this paper, we present a novel technique, *boundary postprocessing,* which can help to achieve significant memory and communication savings. The idea is motivated by the standard *overlap-add* technique which *first* performs filtering operations on neighboring data blocks independently and completes the computation *later* by summing the partial boundary results together [15]. We extend this idea to the case of multilevel wavelet decompositions using the lifting framework formulated by Daubechies and Sweldens [17]. In the proposed approach, the DWT is modeled as a *finite-state machine*, in which each sample is updated progressively from the initial state (the original data sample) to the final state (the wavelet coefficient) with the help of samples in its local neighborhood. Obviously, samples near data boundaries cannot be fully updated due to lack of data from neighboring blocks. Rather than leaving them unchanged, as was done in previous approaches, we propose to update these samples into intermediate states and preserve these partially transformed results (state information) for later processing. The FSM model thus ensures that correct transform can still be achieved for these boundary samples using the preserved state information. Because of the partial computation and the state preservation, we will show that the buffer size in sequential algorithms and the communication overhead in parallel algorithms can be reduced.

Some recent works have also explored (independently of our work) the use of lifting factorizations for memory savings in sequential DWT implementations [19]–[21]. The novelty of our work is that, first, we introduce partial computations for boundary samples at multiple decomposition levels for memory savings and second, we propose that processors exchange data after multilevel decompositions for communication savings. Application of the proposed *boundary postprocessing* technique results in two new DWT system architectures, the *overlap-state* sequential, and the *split-and-merge* parallel. We will show how the proposed technique can be used to reduce the memory requirement and the interprocessor communication overhead in the architecture designs.

We mention that, throughout this paper, we focus on the Mallat tree-structured [22] multilevel octave-band wavelet decomposition system with critical sampling using a two-channel wavelet filterbank. The extensions of our work to other DWT systems, including *standard* DWTs [23], multichannel wavelet filterbank, and wavelet packet decompositions are straightforward. The rest of the paper is organized as follows. In the next section, the *finite state machine* model is introduced for the DWT and the *boundary postprocessing* technique for the transform near block boundaries is presented. The proposed sequential and parallel architectures are given in Sections III and IV, respectively, along with performance analysis and experimental results. Section V concludes our work.

II. Finite-State Machine Model for DWT

In this section, we first introduce the FSM model for DWT based on the lifting factorization. Then we discuss a postprocessing technique for DWT computation near block boundaries.

A. The Finite-State Machine Model

The polyphase matrix $\mathbf{P}(z)$ of any FIR wavelet filterbank has a factorization form ([17, Theorem 7]) as

$$\mathbf{P}(z) = \prod_{i=1}^{m-1} \begin{bmatrix} 1 & s^i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t^i(z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & \frac{1}{K} \end{bmatrix} \tag{1}$$

where $(s^i(z), t^i(z))$ are Laurent polynomials, which are called *prediction* and *updating* operations, respectively. Without loss of generality, we use $\mathbf{e}^i(z)$ to represent the elementary matrices. That is

$$\mathbf{e}^i(z) \equiv \begin{bmatrix} 1 & s^i(z) \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad \mathbf{e}^i(z) \equiv \begin{bmatrix} 1 & 0 \\ t^i(z) & 1 \end{bmatrix}.$$

Let us consider the time domain filtering operations corresponding to $\mathbf{e}^i(z)$. By definition, we have $h(z) = h_e(z^2) +$

$z^{-1}h_o(z^2)$ where $(h_e(z), h_o(z))$ are the two polyphase components of filter $h(z)$. Each $\mathbf{e}^i(z)$ corresponds to two time domain filters

$$\begin{cases} h^i(z) = 1 + z^{-1}s^i(z^2) \\ g^i(z) = z^{-1} \end{cases} \quad \text{or} \quad \begin{cases} h^i(z) = 1 \\ g^i(z) = t^i(z^2) + z^{-1} \end{cases} \tag{2}$$

where $h^i(z)$ and $g^i(z)$ are the low- and high-pass filters in the analysis filterbank, respectively. In time domain, this corresponds to In time domain, this corresponds to

$$\begin{bmatrix} x_e^{i+1}(n) \\ x_o^{i+1}(n) \end{bmatrix} = \begin{bmatrix} \sum_k h^i(k)x^i(2n-k) \\ \sum_k g^i(k)x^i(2n-k) \end{bmatrix}. \tag{3}$$

For a N-point input sequence (assume N is even), $x_e^i(n) = x^i(2n)$ and $x_o^i(n) = x^i(2n+1)$. The state transition in vector form for the upper triangular elementary matrix (similar form for the lower triangular elementary matrix) is

$$\begin{bmatrix} x^{i+1}(0) \\ x^{i+1}(1) \\ x^{i+1}(2) \\ x^{i+1}(3) \\ \vdots \\ x^{i+1}(2m) \\ x^{i+1}(2m+1) \\ \vdots \\ x^{i+1}(N-1) \end{bmatrix} = \begin{bmatrix} x^i(0) \\ \sum_k g^i(k)x^i(-k) \\ x^i(2) \\ \sum_k g^i(k)x^i(2-k) \\ \vdots \\ x^i(2m) \\ \sum_k g^i(k)x^i(2m-k) \\ \vdots \\ \sum_k g^i(k)x^i(N-2-k) \end{bmatrix}.$$

Obviously, each and every elementary matrix $\mathbf{e}^i(z)$ in the factorization of the polyphase matrix $\mathbf{P}(z)$ can be modeled as such a state transition process.

Consider the input $x(n)$ as a column vector, and define the intermediate states in the process of transformation $\{x^i(n),\ i = 0, 1, \ldots, 2m+1\}$, where $x^i(n)$ is the result of applying the operation $\mathbf{e}^{i-1}(z)$ to $x^{i-1}(n)$, and where the initial input is $x^0(n) = x(n)$. There are two important observations.

1) Every time we generate $x^i(n)$, we only need to store this set of values, i.e., we do not need to know any of the other $x^j(n)$, for $j < i$, in order to compute the output (the final wavelet coefficients).
2) Each sample is updated using samples only from the other polyphase component and itself. Consequently, each sample can be updated *independently*, fully or partially, and written back to its original location, an extended in-place computation feature of the lifting algorithm. For example, the updating of $x^i(1)$ (may only be partially updated because insufficient boundary data is available) will not affect the updating of $x^i(3)$ at this stage.

For the polyphase matrix factorization, this requires that the elementary matrix $\mathbf{e}^i(z)$ can only be in the form of lower/upper triangular matrices with constants on the diagonal entries. This key property of the lifting factorization guarantees that the FSM structure applies. Thus, the wavelet transform based on the polyphase factorization can be modeled as a *finite state machine*, as depicted in Fig. 2, where each elementary matrix $\mathbf{e}^i$ updates the FSM state x^i to the next higher level x^{i+1}. The significance of such a FSM model is that the transform can be stopped at any intermediate stage. As long as the state information is preserved, the computation can be continued at any later time. This is the key reason that one can defer the transform for block boundary samples and still can obtain the correct result.

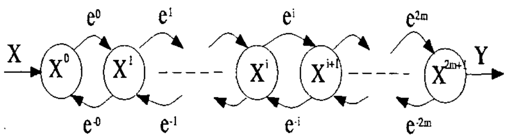
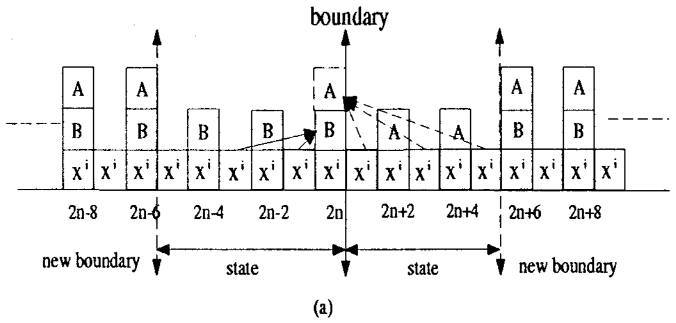


Fig. 2. State transition diagram of DWT as a FSM.



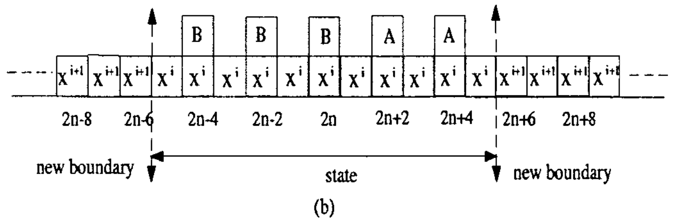


Fig. 3. State transitions across block boundary using $\mathbf{e}^i$. (a) Partial computations near boundaries. (b) After updating, boundary samples stay in intermediate states. The "new boundary" separates fully updated output coefficients from partially computed ones.

B. Boundary Postprocessing

In Fig. 3, we show one *updating* operation for input sequence $x^i(n)$ where even-indexed samples are updated as $x_e^{i+1}(n) = x^{i+1}(2n) = \sum_k h^i(k)x^i(2n-k)$. Denote $H^i(z) = \sum_{n=-a^i}^{b^i} h^i(n)z^{-n}$ $(a^i > 0, b^i > 0)$, then

$$x^{i+1}(2n) = \underbrace{\sum_{k=-a^i}^{-1} h^i(k)x^i(2n-k)}_{A(2n)} + x^i(2n) + \underbrace{\sum_{k=1}^{b^i} h^i(k)x^i(2n-k)}_{B(2n)}$$

where $A(2n)$ and $B(2n)$ are respectively the contributions from the anticausal and causal part of filter h.

Now let us consider the computations near the block boundary at point $2n$. Take sample $x^i(2n)$, for example. It is obvious that it cannot be updated into state $i+1$ because the anticausal filtering result $A(2n)$ cannot be computed due to lack of data samples from the other block, i.e., samples $\{x^i(2n+1), x^i(2n+3), x^i(2n+5)\}$, as shown in Fig. 3(a).

A typical approach used in most existing DWT algorithms is to buffer $x^i(n)$ and wait until the future samples are available, see, e.g., [3], [1], [4]. This direct buffering approach, however, forces one also to buffer whatever samples are necessary in the current block for the computation of $B(2n)$, i.e., samples $\{x^i(2n-3), x^i(2n-1)\}$ in this case. Clearly, for the future updating of sample $x^i(2n)$, the buffer should at least be large enough to hold three samples.

Observing the fact that $B(2n)$ is readily available (all samples needed are in the same block as $x^i(2n)$), we propose to first update $x^i(2n)$ into $x^i(2n) + B(2n)$ and then buffer this partially computed result at the same location of $x^i(2n)$. Notice that $x^i(2n)$ can be over-written (i.e., in-place computation) because no samples in the current block or the other block need the *original* value $x^i(2n)$ for their updating operations as described before. As soon as samples $\{x^i(2n+1), x^i(2n+3), x^i(2n+5)\}$ become available, $A(2n)$ can be computed and $x^i(2n)$ can be updated into $x^{i+1}(2n) = x^i(2n) + A(2n) + B(2n)$. Clearly, in this case we only need a one sample size buffer to store the partially computed result at sample location $2n$ for the future updating. Thus compared to the case of direct buffering of $x^i(2n)$, buffering partially transformed coefficients helps to reduce the memory requirement.

The same analysis also applies for other samples at both sides of the block boundary. The end state after the application of $\mathbf{e}^i(z)$ is shown in Fig. 3(b). As one can see, the physical boundary splits into two and extends inwards in both blocks. The next stage state transition will operate against these two new boundaries. Notice that locations of these two new boundaries can be derived easily using the filtering tap information a^i and b^i before the stage i updating operation. As a matter of fact, given the lifting factorization of the polyphase matrix $\mathbf{P}(z)$, one knows exactly the end state number of each sample in the input data sequence. Therefore, the state number of each sample needs not to be stored for future processing.

Consequently, the buffer size at stage i, B^i, is only determined by the number of samples that need to be stored. For given filters $(h^i(z), g^i(z))$, $B^i = l^i - 2$ where l^i is the number of taps of the longest of the two filters. Assuming a total of m state transitions, then the total buffer size is $B = \sum_{i=0}^{m-1} B^i$. For the (9,7) filters using the factorization given in [17], one can obtain $l^0 = l^1 = l^2 = l^3 = 3$ and thus $B = 4$. That is, only four partially transformed samples need to be preserved rather than seven samples as before. We emphasize that the buffer size reduction is based on a typical assumption in in-place lifting algorithms that each sample (including original data samples and partially transformed samples) needs the same amount of storage space. Since lifting factorization of a given polyphase matrix is not unique, one would choose the factorization which gives the minimum B if the amount of memory is limited.

We thus call this approach of preserving intermediate state information and completing the transform later the *boundary postprocessing* technique. Fig. 4 shows an example three-level wavelet decomposition using the proposed technique. Compared to the approach shown in Fig. 1, one can see that, only one data exchange is necessary between the two blocks. The amount of data exchanged is also reduced due to the partial computation as described above.

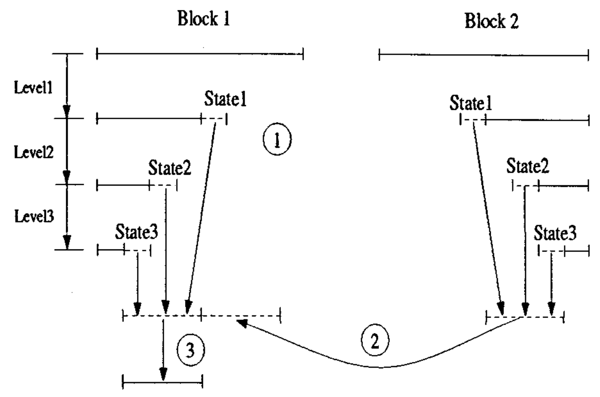


Fig. 4. Example dataflow chart of a three-level wavelet decomposition using the proposed *boundary postprocessing* technique. Solid lines: completely transformed data. Dashed lines: partially transformed data. Operation 1: each block transforms its own allocated data independently and state information is buffered. Operation 2: state information is communicated to neighboring blocks. Operation 3: complete transform for the boundary data samples.

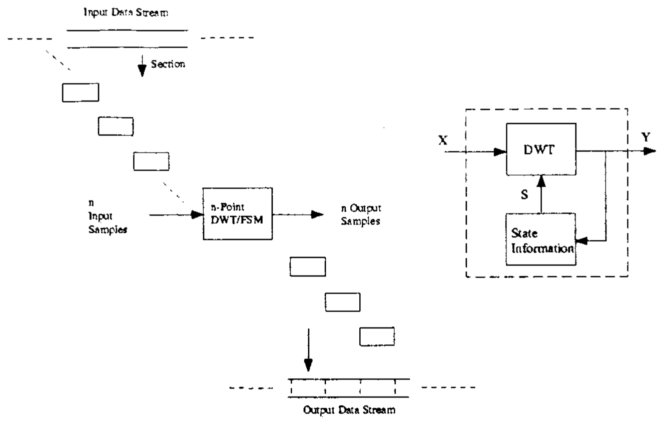


Fig. 5. The proposed sequential architecture for DWT.

III. Overlap-State Sequential Architecture

In Fig. 5, the proposed *overlap-state* sequential DWT system architecture is shown. The input data sequence is segmented into nonoverlapping blocks of length N and fed into the DWT/FSM one block at a time. The state information is overlapped between consecutive blocks. The computed wavelet coefficients are concatenated together to give the final result.

As shown in the last section, the memory requirement depends on the specific lifting factorization used in the implementation. Table I provides memory requirements for commonly used wavelet filterbanks, including the Daubechies (9,7) filters, the (2,10) filters, and the cubic B-Splines CDF(4,2) filters; their factorizations can be found in [24]. A direct extension of 1-D DWT becomes the strip-sequential 2-D DWT as shown in Fig. 6, where the input data is transformed one strip at a time with state information overlapped only vertically (boundary extensions are used horizontally). The buffer size B_s for the state information in a J-level decomposition can be computed as

$$B_s = \sum_{j=0}^{J-1} W W_{c1}^{j} 2^{-j} \tag{4}$$

TABLE I
COMPARISON OF MEMORY REQUIREMENTS IN 1-D DWT
N: SEQUENCE LENGTH. L: FILTER LENGTH. J: DECOMPOSITION LEVEL.
B: NUMBER OF PARTIALLY COMPUTED SAMPLES AT EACH LEVEL

	SSWT[16]	*RPA[3]*	*Proposed*
L-tap	$N+(2^J-1)(L-2)$	$N+J(L-2)$	$N+JB$
(9,7)	$N+7(2^J-1)$	$N+7J$	$N+4J$
(2,10)	$N+8(2^J-1)$	$N+8J$	$N+4J$
CDF(4,2)	$N+5(2^J-1)$	$N+5J$	$N+3J$

TABLE II
COMPARISON OF MEMORY REQUIREMENTS IN LINE-BASED SYSTEMS
W: ROW WIDTH. B: NUMBER OF PARTIALLY COMPUTED SAMPLES AT EACH LEVEL. $\alpha = (2^J - 1)$. $\beta = (1 - 2^{-J})$

	SSWT[16]	*RPA[3]*	*Proposed*
L-tap	$W\alpha(L-2)$	$2W\beta(L-2)$	$2W\beta B$
(9,7)	$7W\alpha$	$14W\beta$	$8W\beta$
(2,10)	$8W\alpha$	$16W\beta$	$8W\beta$
CDF(4,2)	$5W\alpha$	$10W\beta$	$6W\beta$

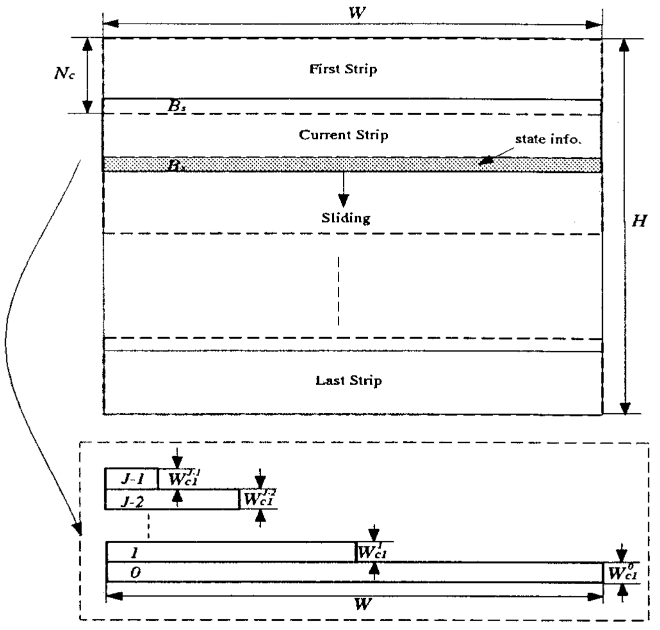


Fig. 6. The strip-sequential 2-D DWT system on data of size $W \times H$. The input is transformed one strip at a time from top to bottom. State information is overlapped vertically only.

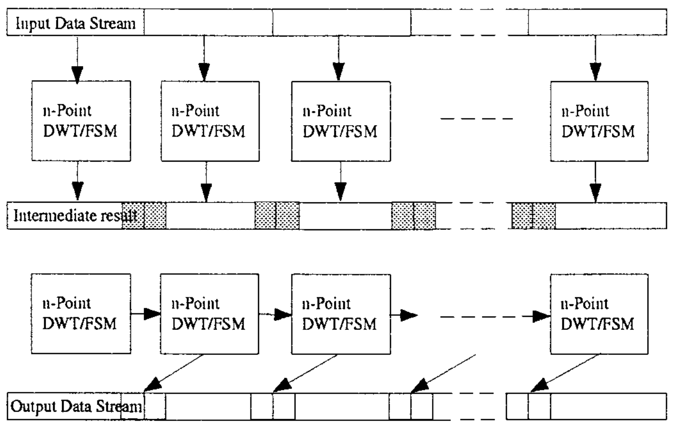


Fig. 7. Proposed parallel architecture for DWT.

where W is the data width W_{c1}^j (subscript "$c1$" stands for bottom side of the data strip) is the number of rows partially computed at level j.

A special case of this strip-sequential 2-D DWT system is the line-based DWT system described in [1], [10], which assumes all completely transformed coefficients are not buffered. Based on this assumption, for one level decomposition using a L-tap filterbank, only $L-2$ rows need to be buffered in SSWT or RPA (if counting the two new input rows, the total buffer size should be L). The memory requirement in the proposed system is always upper-bounded by that of RPA and can have substantial reduction (approximately 40%) for commonly used wavelet filterbanks, as shown in Table II. For example, using the (9,7) filters, the proposed system only needs to buffer four rows (i.e., $W_{c1}^j = 4$) of data at each level while RPA needs to buffer 7 rows of data. The constant $\beta = (1 - 2^{-J})$ is due to line downsampling at each decomposition level. Refer to [24] for more details on memory requirements and other 2-D DWT systems (e.g., the block sequential system).

IV. SPLIT-AND-MERGE PARALLEL ARCHITECTURE

In Fig. 7, the proposed parallel DWT architecture is shown. The input data is uniformly segmented into nonoverlapping blocks and allocated onto p available processors. Each processor computes its own allocated data up to the required wavelet decomposition level. This stage is called *Split*. The output from this stage consists of: 1) completely transformed coefficients and 2) the state information (partially updated boundary samples). In the second stage, *Merge*, a one-way communication is initiated and the state information is transfered to the neighboring processors. The state information from the neighbor processor is then combined together with its own corresponding state information to complete the whole DWT transform. This is further illustrated by Fig. 8, where it can be seen how the operations in each processor are carried as far as possible in the split operation, while in the merge operation the processor will combine the available information to update the partially computed outputs.

A. Communication Delay

The communication delay is the time used for communicating data between adjacent processors. Let t_{setup} be the communication setup time, e.g., the handshake time in an asynchronous communication protocol. For a J level wavelet decomposition, the total communication D_{old} [4] is

$$D_{\text{old}} = J(t_{\text{setup}} + (L-2)t_c) \tag{5}$$

where t_c is the time to transfer one data sample and $(L-2)$ is the number of boundary samples exchanged to adjacent processor at each level. In the proposed approach, however, only one communication setup is necessary to communicate the state information between adjacent processors. Furthermore, the size of the state information at each decomposition level B is upper

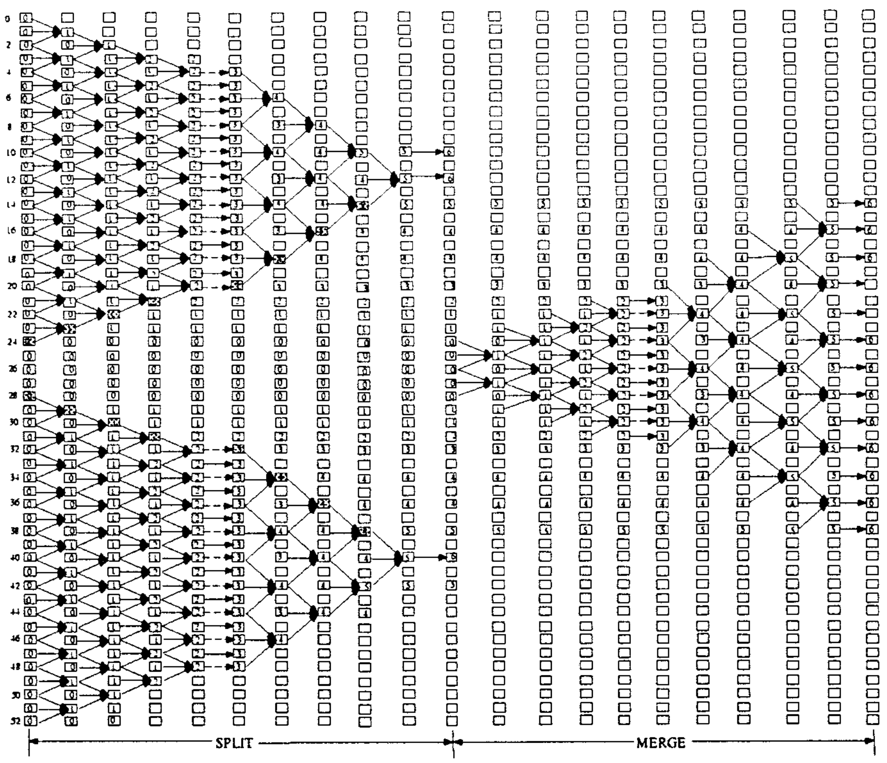


Fig. 8. An example *split-and merge* parallel DWT architecture using the Daubechies (9, 7) filters for two level decompositions. Shaded boxes represent partially updated samples to be exchanged between processors in the *merge* stage. Notice that samples {25, 26, 27} are left unprocessed for clarity and the block boundary can be at any one of these three samples.

bounded by $(L-2)$ [24]. So the upper bound of the communication delay D_{new} is

$$D_{\text{new}} \leq t_{\text{setup}} + J(L-2)t_c. \tag{6}$$

As one can see, using the *boundary postprocessing* approach, the communication overhead for link setup is reduced. Notice that the comparison is based on the assumption that all data samples which have to be buffered and exchanged (be it an original data sample or a partially computed intermediate sample) require the same storage space. This is a reasonable assumption when the in-place lifting algorithm is used to compute the wavelet transform. The reduction of the communication time certainly contributes to the total DWT computation time reduction. However, we mention that the overall contribution depends on the relative weight of the communication overhead in the total DWT computation. In general, more gain can be achieved for parallel systems with slow communication links.

To test the efficiency of the proposed parallel architecture, four different DWT algorithms are implemented using two SUN ULTRA-1 workstations running the LAM 6.1 parallel platform developed in Ohio Supercomputer Center [14]. The algorithms compared are the sequential lifting algorithm, parallel standard algorithm (subsample-filtering approach), parallel lifting scheme, and our proposed parallel scheme. The test wavelet filters are the (9,7) filters and the test image is the Lena grayscale image of size 512×512. A 2-D separable wavelet transform is implemented with strip data partition [4] between processors (refer to Fig. 6). We implemented all the algorithms ourselves with no specific optimizations on the codes except for the compiler optimization. Further improvement of algorithms performances is certainly possible through more rigorous code optimizations. To compare the performances, the relative speedup $T_{\text{old}}/T_{\text{new}} - 1$ is computed and the averaged results over 50 running instances for 1–5 decomposition levels are given in Table III.

TABLE III
DWT Running Time and Speedup of Different Parallel Algorithms (in Seconds). S: Sequential. P: Parallel. L: Lifting

J	SL	PStandard		PLifting		PProposed	
		t	s	t	s	t	s
1	0.364	0.311	17%	0.274	33%	0.204	78%
2	0.365	0.327	11%	0.290	26%	0.234	56%
3	0.395	0.349	13%	0.294	34%	0.237	67%
4	0.403	0.351	15%	0.304	34%	0.238	69%
5	0.404	0.367	9%	0.316	28%	0.242	67%

It can be seen from the results that our proposed parallel algorithm can significantly reduce the DWT computation time even compared with the fastest available parallel algorithm, i.e., the parallel lifting algorithm. Notice that the improvement is not linear with the increase of the decomposition level. The reason is that, though the communication overhead increases with the decomposition level, the total numerical computation also increases. Another observation is that the improvement of the proposed algorithm at one level decomposition is even greater than that at multiple level decompositions. It suggests that, in our experimental system setup, the gain due to saved amount of exchanged data is greater than that due to saved number of communication setups, though, future work is needed to further investigate this issue.

V. Conclusions

To conclude, we have proposed a new *boundary postprocessing* technique for the DWT computation near block boundaries. Performance analysis and experimental results show that the auxiliary buffer size for boundary DWT and the communication overhead can be significantly reduced by using the proposed technique. The results presented here can be easily extended to 2-D or higher dimensional wavelet transforms by using separable transform approaches [24].

References

[1] C. Chrysafis and A. Ortega, "Line based, reduced memory, wavelet image compression," in *Proc. Data Compression Conf.*, 1998, pp. 398–407.

[2] O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Trans. Inform. Theory*, vol. 38, pp. 569–586, Mar. 1992.

[3] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. Signal Processing*, vol. 42, pp. 673–676, Mar. 1994.

[4] J. Fridman and E. S. Manolakos, "On the scalability of 2-D discrete wavelet transform alogrithms," *Multidimensional Syst. Signal Processing*, no. 8, pp. 185–217, 1997.

[5] M. A. Trenas, J. Lopez, and E. L. Zapata, "A memory system supporting the efficient SIMD computation of the two dimensional DWT," in *Proc. ICASSP*, 1998, pp. 1521–1524.

[6] W. Jiang and A. Ortega, "A parallel architecture for DWT based on lifting factorization," in *Proc. SPIE Parallel and Distributed Methods for Image Processing III*, vol. 3817, Oct. 1999, pp. 2–13.

[7] A. Ortega, W. Jiang, P. Fernandez, and C. Chrysafis, "Implementations of the discrete wavelet transform: complexity, memory, and parallelization issues," in *Proc. SPIE: Wavelet Applications in Signal and Image Processing VII*, vol. 3813, Oct. 1999, pp. 386–400.

[8] W. Jiang and A. Ortega, "Efficient DWT system architecture design using filterbank factorizations," in *Proc. ICIP*, vol. 2, Oct. 1999, pp. 749–753.

[9] *Proc. IEEE: Special Issue on Wavelets*, no. 4, 1996.

[10] C. Chrysafis and A. Ortega, "Line based, reduced memory, wavelet image compression," *IEEE Trans. Image Processing*, vol. 9, pp. 378–389, May 2000.

[11] J. Bowers, L. Keith, N. Aranki, and R. Tawel, "IMAS integrated controller electronics," Jet Propulsion Laboratory, Pasadena, CA, 1998.

[12] L. Yang and M. Misra, "Coarse-grained parallel algorithms for multidimensional wavelet transforms," *J. Supercomputing*, vol. 12, no. 1/2, pp. 99–118, 1998.

[13] T. E. Anderson, D. E. Culler, and D. A. Patterson, "A case for NOW (Networks of Workstations)," *IEEE Micro*, vol. 151, pp. 54–64, Feb. 1995.

[14] LAM/MPI parallel computing. University of Notre Dame, Notra Dame, IN. [Online]. Available: http://www.mpi.nd.edu/lam

[15] R. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley, 1985.

[16] F. Kossentini, "Spatially segmented wavelet transform," UBC, ISOIEC JTC 1SC29WG1 WG1N868, 1998.

[17] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 247–269, 1998.

[18] "Report on core experiment codeff1 Complexity reduction of SSWT," Motorola Australia, UBC, ISOIEC JTC 1SC29WG1 WG1N881, 1998.

[19] C. Chrysafis, "Low memory line-based wavelet trasform using lifting scheme," HP Labs, ISOIEC JTC 1SC29WG1 WG1N978, 1998.

[20] P. Onno, "Low memory line-based wavelet transform using lifting scheme," Canon Research Center France, ISOIEC JTC 1SC29WG1 WG1N1013, 1998.

[21] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, "Optimal memory organization for scalable texture codecs in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 218–243, Mar. 1999.

[22] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 674–693, 1989.

[23] G. Beylkin, R. Coifman, and V. Rokhlin, "Fast wavelet transforms and numerical algorithms I," *CPAM*, vol. 44, pp. 141–183, 1991.

[24] W. Jiang and A. Ortega, "Discrete wavelet transform system architecture design using filterbank factorization," Univ. Southern California, Los Angeles, CA, 1999.